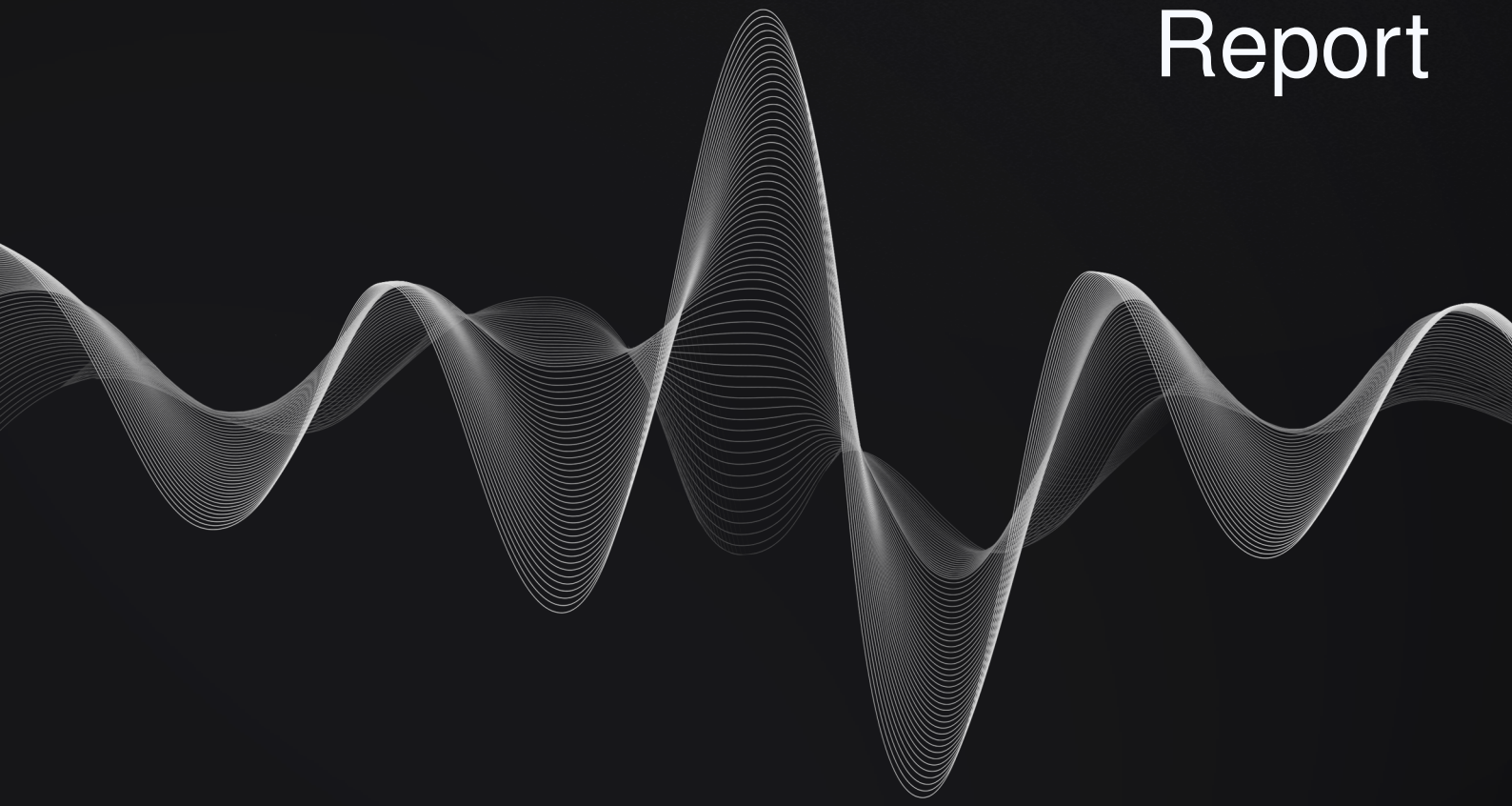
RESONANCE

Chiliz Chain

Fan Token Staking Smart Contract Audit Report



Blockchain, Emerging Technology, and Web2
CYBERSECURITY PRODUCT & SERVICE ADVISORY

Document Control

PUBLIC **FINAL**(v2.2)

Audit_Report_CHLZ-STK_FINAL_22

Jul 1, 2025	v0.1	Luis Arroyo: Initial draft
Jul 2, 2025	v0.2	Luis Arroyo: Added findings
Jul 3, 2025	v1.0	Charles Dray: Approved
Jul 11, 2025	v1.1	Luis Arroyo: Reviewed findings
Jul 16, 2025	v1.2	Luis Arroyo: Added findings
Jul 21, 2025	v1.3	Luis Arroyo: Reviewed findings
Jul 30, 2025	v2.0	Charles Dray: Finalized
Aug 20, 2025	v2.1	Charles Dray: Published
Oct 17, 2025	v2.2	João Simões: Updated title

Points of Contact	James Scicluna	Chiliz Chain	james.scicluna@chiliz.com
	Charles Dray	Resonance	charles@resonance.security
Testing Team	Luis Arroyo	Resonance	luis.arroyo@resonance.security
	João Simões	Resonance	joao@resonance.security
	Michał Bazyli	Resonance	michal@resonance.security

Copyright and Disclaimer

Contents

Executive Summary

Chiliz Chain contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between June 30, 2025 and July 03, 2025. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 2 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 4 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

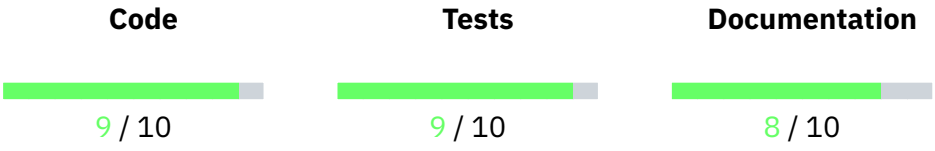
The ultimate goal of the audit was to provide Chiliz Chain with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.

System Overview

The Staking3 protocol manages staking, locking, and unstaking of fan tokens. It also allows users to upgrade their staked tokens (e.g., from a non-divisible to a divisible version).

The FTSwap allow users to exchange their old tokens for new approved ones and ensures only the staking contract can perform these swaps on behalf of users' staked balances.

Repository Coverage and Quality



Resonance's testing team has assessed the Code, Tests, and Documentation coverage and quality of the system and achieved the following results:

- The code follows development best practices and makes use of known patterns, standard libraries, and language guides. It is easily readable and uses the latest stable version of relevant components. Overall, **code quality is excellent**.
- Unit and integration tests are included. The tests cover both technical and functional requirements. Code coverage is 100%. Overall, **tests coverage and quality is excellent**.
- The documentation includes the specification of the system, technical details for the code, relevant explanations of workflows and interactions. Overall, **documentation coverage and quality is good**.

Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: `chiliz-chain/fan-token-swap-smart-contract`
- Hash: 1ab4ab31280fcfb6c6c30f60ba36e1beed74ac65
- Repository: `chiliz-chain/fan-token-staking`
- Hash: a7a1ec7665a8c15c76ba203969d8a3d421bfbfeb

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- Financial related attacks

Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

Source Code Review - Solidity EVM

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Solidity EVM audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Reentrancy attacks
- Frontrunning attacks
- Unsafe external calls
- Unsafe third party integrations
- Denial of service
- Access control issues

- Inaccurate business logic implementations
- Incorrect gas usage
- Arithmetic issues
- Unsafe callbacks
- Timestamp dependence
- Mishandled panics, errors and exceptions

Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

Impact \ Likelihood	Very Likely	Likely	Unlikely
Strong	Critical	High	Medium
Medium	High	Medium	Low
Weak	Medium	Low	Info

Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.

Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

- **"Quick Win"** Requires little work for a high impact on risk reduction.
- **"Standard Fix"** Requires an average amount of work to fully reduce the risk.
- **"Heavy Project"** Requires extensive work for a low impact on risk reduction.

RES-01	Cooldown For Stakes Can Be Bypassed Or Increased		Acknowledged
RES-02	Decimals Calculation May Be Wrong		Resolved
RES-03	Missing Initializer Call For Parent Contract		Resolved
RES-04	Missing Zero Address Validation On setFtStaking()		Resolved
RES-05	Missing Zero Address Validation On setFtSwap()		Resolved
RES-06	Missing Limits In setUnstakePeriod()		Resolved
RES-07	_swapWithBurnAndMint Allows 0 Amount Swaps		Resolved
RES-08	newFTAmount Is Not Emitted		Resolved
RES-09	Unnecessary Initialization Of Variables With Default Values		Resolved
RES-10	Missing Reentrancy Guard Modifier On swapStakedTokens() And swapStaked()		Resolved
RES-11	Mixed Usage Of Initializing Functions In initialize()		Resolved
RES-12	Floating Pragma		Resolved
RES-13	Incorrect Initialization Order On initialize()		Resolved
RES-14	Test Cases Not Working Properly		Resolved

Cooldown For Stakes Can Be Bypassed Or Increased

High | **RES-CHLZ-STK01** | Data Validation | **Acknowledged**

Code Section

- Staking3.sol#L365

Description

The protocol adds a feature where tokens can have different cooldown periods. This may mean that in some cases, users can bypass this period by swapping tokens.

It is possible that the new token will have a lower cooldown than the old token, thus allowing previously unstaked tokens to have a shorter cooldown period.

These are the steps to replicate this issue:

1. Stake old tokens (asuming 1000 seconds of locking period).
2. Wait less time than `unstakePeriodPerToken` of the old token and unstake them.
3. Swap tokens.
4. The lock time for your position has decreased depending on the `unstakePeriodPerToken` of the new token. For example, if this value is 1 second, then the remaining time is bypassed.

Recommendation

It is recommended to add a custom remaining time for each cooldown peroid in the new swapped tokens equal to the remainder of the old tokens, so that users can swap tokens without affecting the previous locks.

Status

The issue was acknowledged by Chiliz team. The development team stated "The swapStakedTokens() function is designed to swap all eligible v1 tokens, restaking only the v2 equivalent portion that was previously staked (totalUnstakable). V1 tokens from other states (totalClaimable, totalPendingUnstake) are moved to the user's wallet, bypassing the cooldown period, and any v2 tokens unstaked after the swap are subject to a full cooldown period.".

Decimals Calculation May Be Wrong

Medium | RES-CHLZ-STK02 | Data Validation | Resolved

Code Section

- FTSwap.sol#L154

Description

In the FTSwap protocol, the owner updates or adds new pairs. In case any of these old Fan Tokens have decimals, the calculation of the new Fan Tokens would be incorrect, since the formula used would return fewer tokens than expected.

Suppose:

- `oldFT` has **2 decimals** (unexpected).
- `newFT` has **18 decimals**.

If a user swaps **1** `oldFT` token, the calculation is:

```
newFTAmount = 1 * 10**18 / 10**2 = 1 * 1e18 / 100 = 1e16
```

So the user receives **0.01** of the intended amount (1e16 instead of 1e18).

Recommendation

Before performing the calculation for `newFTAmount`, explicitly check that `oldFT` has 0 decimals to prevent incorrect swap rates due to unexpected token configurations.

Status

The issue has been fixed in 35cade3b5444bce5f27e11f1d85afd110ff3b970.

Missing Initializer Call For Parent Contract

Medium | RES-CHLZ-STK03 | Data Validation | Resolved

Code Section

- Staking3.sol#L146

Description

The Staking3 contract does not call `__ReentrancyGuard_init()` in the `initialize()` function, which means that the internal reentrancy status variable will not be set. This can lead to the `nonReentrant` modifier not blocking reentrant calls, leaving the contract vulnerable to reentrancy attacks or, in some cases, functions with `nonReentrant` may revert unexpectedly because the guard is in an uninitialized state.

Recommendation

It is recommended to add the `__ReentrancyGuard_init();` call to the initialize function.

Status

The issue has been fixed in ef0ef860cfa2e694a54eae55b36c0d332d46e335.

Missing Zero Address Validation On setFtStaking()

Low | **RES-CHLZ-STK04** | Data Validation | **Resolved**

Code Section

- FTSwap.sol#L61

Description

Input parameter in `setFtStaking()` function is not being validated against the Zero Address.

Mistakes can be made by the owner of the smart contract, allowing for unwanted transactions to take place in the future.

Recommendation

It is recommended to perform a validation against the Zero Address to ensure proper variable values are handled properly and successfully.

Status

The issue has been fixed in 35cade3b5444bce5f27e11f1d85afd110ff3b970.

Missing Zero Address Validation On setFtSwap()

Low | **RES-CHLZ-STK05** | Data Validation | **Resolved**

Code Section

- Staking3.sol#L220

Description

Input parameter in `setFtSwap()` function is not being validated against the Zero Address.

Mistakes can be made by the owner of the smart contract, allowing for unwanted transactions to take place in the future.

Recommendation

It is recommended to perform a validation against the Zero Address to ensure proper variable values are handled properly and successfully.

Status

The issue has been fixed in ef0ef860cfa2e694a54eae55b36c0d332d46e335.

Missing Limits In setUnstakePeriod()

Low | **RES-CHLZ-STK06** | Data Validation | **Resolved**

Code Section

- Staking3.sol#L250

Description

The `setUnstakePeriod()` function does not have an upper limit of the seconds needed to claim tokens. This may affect the user experience and it may not be practical.

Recommendation

It is recommended to add a check to ensure that `unstakePeriod` is correct and is in a suitable range.

Status

The issue has been fixed in 47921810f2977531d0ae1466d859b6f3202753e9.

_swapWithBurnAndMint Allows 0 Amount Swaps

Info	RES-CHLZ-STK07	Data Validation	Resolved

Code Section

- FTSwap.sol#L147

Description

The function does not check if `oldFTAmount` is zero. This could allow zero-amount swaps, which may cause unnecessary events and state changes.

Recommendation

It is recommended to add a check to ensure `oldFTAmount > 0`.

Status

The issue has been fixed in 35cade3b5444bce5f27e11f1d85afd110ff3b970.

newFTAmount Is Not Emitted

Info | RES-CHLZ-STK08 | Code Quality | Resolved

Code Section

- FTSwap.sol#L157

Description

The `Swapped` event in `_swapWithBurnAndMint()` fucntion does not include `newFTAmount`. This may affect potential off-chain accounting.

Recommendation

Emit the `newFTAmount` parameter if possible.

Status

The issue has been fixed in 35cade3b5444bce5f27e11f1d85afd110ff3b970.

Unnecessary Initialization Of Variables With Default Values

Info | **RES-CHLZ-STK09** | Gas Optimization | **Resolved**

Code Section

- Staking3.sol#L91
- Staking3.sol#L123
- Staking3.sol#L475
- Staking3.sol#L481
- Staking3.sol#L519
- Staking3.sol#L607

Description

In the Solidity programming language, all variables are automatically initialized to a default value corresponding to their type when they are declared. For example, integer types are initialized to `0`, boolean types to `false`, and address types to `0x0000000000000000000000000000000000000000`. Explicitly initializing variables to these default values when they are declared is therefore redundant, and since each operation in a contract costs gas, it results in unnecessary gas costs. This could potentially impact the contract's efficiency and the cost of executing its functions.

Several instances of this issue are found across the code base.

Recommendation

It is recommended to review the smart contract's code for variable declarations where variable are being explicitly initialized to the type's default value.

Status

The issue has been fixed in ef0ef860cfa2e694a54eae55b36c0d332d46e335.

Missing Reentrancy Guard Modifier On swapStakedTokens() And swapStaked()

Info	RES-CHLZ-STK10	Data Validation	Resolved

Code Section

- Staking3.sol#L333
- FTSwap.sol#L106

Description

The `swapStakedTokens()` function unstakes, claims, swap tokens transfers them back from user to stake the new token. The `swapStaked()` function calls burn and mint functions in `_swapWithBurnAndMint()`. As these functions transfer tokens to user and call external contracts, it is important to use the `nonReentrant` modifier in the same way as in the rest of the protocol.

Recommendation

It is recommended to add the `nonReentrant` modifier to the mentioned function to protect against reentrancy vulnerabilities

Status

The issue has been fixed in ef0ef860cfa2e694a54eae55b36c0d332d46e335 and 35cade3b5444bce5f27e11f1d85afd110ff3b970.

Mixed Usage Of Initializing Functions In initialize()

Info | **RES-CHLZ-STK11** | Code Quality | **Resolved**

Code Section

- Staking3.sol#L146

Description

The `initialize()` function in the `Staking3` contract uses both internal and external functions to set roles `grantRole()` and `_grantRole()`. Using both works, but is not necessary. Using only `_grantRole` is simpler and more efficient for setup, as it is possible to avoid unnecessary access checks during initialization.

Recommendation

For initialization, it is recommended to use `_grantRole()` for all initial role assignments.

Status

The issue has been fixed in ef0ef860cfa2e694a54eae55b36c0d332d46e335.

Floating Pragma

Info | **RES-CHLZ-STK12** | Code Quality | **Resolved**

Code Section

- Staking3.sol#L2
- FTSwap.sol#L2

Description

The project uses floating pragmas `^0.8.23`.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Recommendation

It is recommended to use a strict and locked pragma version for solidity code. Preferably, the version should be neither too new or too old.

Status

The issue has been fixed in ef0ef860cfa2e694a54eae55b36c0d332d46e335 and 35cade3b5444bce5f27e11f1d85afd110ff3b970.

Incorrect Initialization Order On initialize()

Info | **RES-CHLZ-STK13** | Code Quality | **Resolved**

Code Section

- FTSwap.sol#L37-L42
- Staking3.sol#L146-L154

Description

According to best practices, the inherited smart contracts should be initialized according to their inheritance order, from the most base-like to the most derived.

Recommendation

It is recommended to follow the same order of inheritance and initialization of smart contracts to follow best practices.

Status

The issue has been fixed in ef0ef860cfa2e694a54eae55b36c0d332d46e335 and 35cade3b5444bce5f27e11f1d85afd110ff3b970.

Test Cases Not Working Properly

Info | RES-CHLZ-STK14 | Code Workflow | Resolved

Code Section

- Staking3.t.sol#L365

Description

The `ERC20BurnableMintable` contract located in the `Staking3.t.sol` does not implement the `burn()` function. The tests that use this function will not work properly.

Recommendation

It is recommended to add the `burn()` function in the mock token to ensure that all the needed functions are present.

Status

The issue has been fixed in 47921810f2977531d0ae1466d859b6f3202753e9.

Proof of Concepts

RES-01 Cooldown For Stakes Can Be Bypassed Or Increased

Added lines:

```
function test_lockTokens() public {
    // stake 15, unstake 6, lock 9
    staking3.stake(15_000_000, address(nonDivisibleToken));
    staking3.unstake(6_000_000, address(nonDivisibleToken));
    skip(501); // skipping half of the time to claim

    // swap tokens
    nonDivisibleToken.approve(address(ftSwap), 15_000_000);
    divisibleToken.approve(address(staking3), 15_000_000 ether);
    staking3.swapStakedTokens(address(nonDivisibleToken));

    // unstake remaining 9, should be locked
    staking3.unstake(9_000_000 ether, address(divisibleToken));
    skip(501); //skipping the rest of the cooldown period

    // cant claim as new unstake peroid is 1000 again
    // token will be locked for 1000 + elapsed time before swap
    vm.expectRevert();
    staking3.claim(address(divisibleToken));
}
```

RES-03 Missing Initializer Call For Parent Contract

When running tests against the mentioned contract, an error is shown at validating initializers.

```
Encountered 1 failing test in test/Staking3.t.sol:Staking3Test
[FAIL: Upgrade safety validation failed:
  src/Staking3.sol:Staking3

      src/Staking3.sol:146: Missing initializer calls for one or more parent
↪  contracts: `ReentrancyGuardUpgradeable`
          Call the parent initializers in your initializer function
          https://zpl.in/upgrades/error-001

FAILED] setUp() (gas: 0)
```